Algorithm Discovery and Design

Chapter 2
The Algorithmic Foundations of Computer Science

Algorithms

- Algorithm
  - Step-by-step method for accomplishing some task
- Operations used to construct algorithms
  - Sequential, Conditional, and Iterative operations
- Why are formal algorithms so important?
  - If we can specify an algorithm to solve a problem, then we can automate its solution
- Computing agent
  - Machine, robot, person, or thing carrying out the steps of the algorithm
Algorithms

• Algorithm
  – Well-ordered collection of unambiguous and effectively computable operations that, when executed, produces a result and halts in a finite amount of time

• Ambiguous statements
  – Go back and do it again (Do what again?)
  – Start over (From where?)

Algorithms (continued)

• Unambiguous operation
  – Can be understood and carried out directly by the computing agent without further simplification or explanation
  – Also called primitive operation

• It is not enough for an operation to be understandable
  – It must also be doable (effectively computable) by the computing agent
Algorithms (continued)

- Algorithm
  - Result must be produced after the execution of a finite number of operations
- Infinite loop
  - Runs forever

Sequential Operations

- Basic sequential operations
  - Computation, input, and output
- Instruction for performing a computation and saving the result
  - Set the value of “variable” to “arithmetic expression”
- Variable
  - Storage location that can hold a data value
Sequential Operations (continued)

• Input operations
  – Submit to the computing agent data values from the outside world that it may then use in later instructions

• Output operations
  – Send results from the computing agent to the outside world

**Figure 2.3** Algorithm for Computing Average Miles per Gallon

<table>
<thead>
<tr>
<th>STEP</th>
<th>OPERATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Get values for gallons used, starting mileage, ending mileage</td>
</tr>
<tr>
<td>2</td>
<td>Set value of distance driven to (ending mileage − starting mileage)</td>
</tr>
<tr>
<td>3</td>
<td>Set value of average miles per gallon to (distance driven ÷ gallons used)</td>
</tr>
<tr>
<td>4</td>
<td>Print the value of average miles per gallon</td>
</tr>
<tr>
<td>5</td>
<td>Stop</td>
</tr>
</tbody>
</table>

*Average Miles per Gallon Algorithm (Version 1)*
Conditional and Iterative Operations

- Sequential algorithm
  - Sometimes called a **straight-line algorithm**
- Control operations
  - **Conditional** and **iterative**
  - Allow us to alter the normal sequential flow of control in an algorithm
- Conditional statements
  - The “question-asking” operations of an algorithm
  - *If/then/else*

![Figure 2.4 The If/Then/Else Pseudocode Statement](image)
Conditional and Iterative Operations (continued)

- **Loop**
  - The repetition of a block of instructions
  - *While* statement

- **Continuation condition**
  - Determines if statement is true or false

- **Infinite loop**
  - Continuation condition never becomes false

**Average Miles per Gallon Algorithm (Version 2)**

<table>
<thead>
<tr>
<th>Step</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Get values for gallons used, starting mileage, ending mileage</td>
</tr>
<tr>
<td>2</td>
<td>Set value of distance driven to (ending mileage – starting mileage)</td>
</tr>
<tr>
<td>3</td>
<td>Set value of average miles per gallon to (distance driven ÷ gallons used)</td>
</tr>
<tr>
<td>4</td>
<td>Print the value of average miles per gallon</td>
</tr>
<tr>
<td>5</td>
<td>If average miles per gallon is greater than 25.0 then</td>
</tr>
<tr>
<td>6</td>
<td>Print the message ‘You are getting good gas mileage’</td>
</tr>
<tr>
<td>7</td>
<td>Else</td>
</tr>
<tr>
<td>8</td>
<td>Print the message ‘You are NOT getting good gas mileage’</td>
</tr>
<tr>
<td></td>
<td>Stop</td>
</tr>
</tbody>
</table>

*Figure 2.5 Second Version of the Average Miles per Gallon Algorithm*
Average Miles per Gallon Algorithm (Version 2)

<table>
<thead>
<tr>
<th>STEP</th>
<th>OPERATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Get values for gallons used, starting mileage, ending mileage</td>
</tr>
<tr>
<td>2</td>
<td>Set value of distance driven to (ending mileage - starting mileage)</td>
</tr>
<tr>
<td>3</td>
<td>Set value of average miles per gallon to (distance driven / gallons used)</td>
</tr>
<tr>
<td>4</td>
<td>Print the value of average miles per gallon</td>
</tr>
<tr>
<td>5</td>
<td>If average miles per gallon is greater than 25.0 then</td>
</tr>
<tr>
<td>6</td>
<td>Print the message “You are getting good gas mileage”</td>
</tr>
<tr>
<td>7</td>
<td>Else</td>
</tr>
<tr>
<td>8</td>
<td>Print the message “You are NOT getting good gas mileage”</td>
</tr>
<tr>
<td>8</td>
<td>Stop</td>
</tr>
</tbody>
</table>

**Figure 2.5 Second Version of the Average Miles per Gallon Algorithm**

**Figure 2.6 Execution of the While Loop**
Conditional and Iterative Operations (continued)

• Loop example

  \textit{Step} \hspace{0.5cm} \textit{Operation}
  \begin{tabular}{ll}
  1 & Set the value of \textit{count} to 1 \\
  2 & While (\textit{count} \leq 100) do step 3 to step 5 \\
  3 & Set \textit{square} to (\textit{count} \times \textit{count}) \\
  4 & Print the values of \textit{count} and \textit{square} \\
  5 & Add 1 to \textit{count} \\
  \end{tabular}

Conditional and Iterative Operations (continued)

• Pretest loop
  – Continuation condition is tested at the \textit{beginning} of each pass through the loop

• Posttest loop
  – Continuation condition is tested at the \textit{end} of the loop body, not the beginning

• Primitives
  – Instructions that computing agent understands and is capable of executing without further explanation
**Average Miles per Gallon Algorithm (Version 3)**

<table>
<thead>
<tr>
<th>STEP</th>
<th>OPERATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( \text{response} = \text{Yes} )</td>
</tr>
<tr>
<td>2</td>
<td>While ( \text{response} = \text{Yes} ) do steps 3 through 11</td>
</tr>
<tr>
<td>3</td>
<td>Get values for gallons used, starting mileage, ending mileage</td>
</tr>
<tr>
<td>4</td>
<td>Set value of distance driven to ((\text{ending mileage} - \text{starting mileage}))</td>
</tr>
<tr>
<td>5</td>
<td>Set value of average miles per gallon to ( \left( \frac{\text{distance driven}}{\text{gallons used}} \right) )</td>
</tr>
<tr>
<td>6</td>
<td>Print the value of average miles per gallon</td>
</tr>
<tr>
<td>7</td>
<td>If average miles per gallon &gt; 25.0 then</td>
</tr>
<tr>
<td>8</td>
<td>Print the message 'You are getting good gas mileage'</td>
</tr>
<tr>
<td>9</td>
<td>Else</td>
</tr>
<tr>
<td>10</td>
<td>Print the message 'You are NOT getting good gas mileage'</td>
</tr>
<tr>
<td>11</td>
<td>If average miles per gallon &gt; 25.0 then</td>
</tr>
<tr>
<td>12</td>
<td>Get a new value for ( \text{response} ) from the user</td>
</tr>
<tr>
<td>13</td>
<td>Stop</td>
</tr>
</tbody>
</table>

**Figure 2.7** Third Version of the Average Miles per Gallon Algorithm

---

**Figure 2.8** Execution of the Do/While Posttest Loop
Examples of Algorithmic Problem Solving

- **Example 1: Go Forth and Multiply**

  *Given 2 nonnegative integer values, $a \geq 0$, $b \geq 0$, compute and output the product $(a \times b)$ using the technique of repeated addition. That is, determine the value of the sum $a + a + a + \ldots + a$ ($b$ times)*
Multiplication of Non-negative Values via Repeated Addition

Get values for \(a\) and \(b\)
If (either \(a = 0\) or \(b = 0\)) then
   Set the value of \(product\) to 0
Else
   Set the value of \(count\) to 0
   Set the value of \(product\) to 0
   While \((count < b)\) do
      Set the value of \(product\) to \((product + a)\)
      Set the value of \(count\) to \((count + 1)\)
   End of loop
   Print the value of \(product\)
   Stop

Figure 2.10 Algorithm for Multiplication of Nonnegative Values via Repeated Addition

Example 2: Looking, Looking, Looking

• Algorithm discovery
  – Finding a solution to a given problem
• Sequential search
  – Standard algorithm for searching an unordered list of values
### Figure 2.11 First Attempt at Designing a Sequential Search Algorithm

<table>
<thead>
<tr>
<th>Step</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Get values for NAME, N₁, ⋯, N₁₀,000 and T₁, ⋯, T₁₀,000</td>
</tr>
<tr>
<td>2</td>
<td>If NAME = N₁ then print the value of T₁</td>
</tr>
<tr>
<td>3</td>
<td>If NAME = N₂ then print the value of T₂</td>
</tr>
<tr>
<td>4</td>
<td>If NAME = N₃ then print the value of T₃</td>
</tr>
<tr>
<td></td>
<td>⋮</td>
</tr>
<tr>
<td>10,000</td>
<td>If NAME = N₉,999 then print the value of T₉,999</td>
</tr>
<tr>
<td>10,001</td>
<td>If NAME = N₁₀,000 then print the value of T₁₀,000</td>
</tr>
<tr>
<td>10,002</td>
<td>Stop</td>
</tr>
</tbody>
</table>

### Figure 2.12 Second Attempt at Designing a Sequential Search Algorithm

<table>
<thead>
<tr>
<th>Step</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Get values for NAME, N₁, ⋯, N₁₀,000 and T₁, ⋯, T₁₀,000</td>
</tr>
<tr>
<td>2</td>
<td>Set the value of i to 1 and set the value of Found to NO</td>
</tr>
<tr>
<td>3</td>
<td>While (Found = NO) do steps 4 through 7</td>
</tr>
<tr>
<td>4</td>
<td>If NAME is equal to the ith name on the list Nᵢ then</td>
</tr>
<tr>
<td>5</td>
<td>Print the telephone number of that person, Tᵢ</td>
</tr>
<tr>
<td>6</td>
<td>Set the value of Found to YES</td>
</tr>
<tr>
<td></td>
<td>Else (NAME is not equal to Nᵢ)</td>
</tr>
<tr>
<td>7</td>
<td>Add 1 to the value of i</td>
</tr>
<tr>
<td>8</td>
<td>Stop</td>
</tr>
</tbody>
</table>
Example 3: Big, Bigger, Biggest

- **Library**
  - Collection of useful algorithms

- **Problem**

  Given a value \( n \geq 1 \) and a list containing exactly \( n \) unique numbers called \( A_1, A_2, \ldots, A_n \), find and print out both the largest value in the list and the position in the list where that largest value occurred.
**Find Largest Algorithm**

Get a value for \( n \), the size of the list
Get values for \( A_1, A_2, \ldots, A_n \), the list to be searched
Set the value of \( \text{largest so far} \) to \( A_1 \)
Set the value of \( \text{location} \) to 1
Set the value of \( i \) to 2
While \( i \leq n \) do
  
  If \( A_i > \text{largest so far} \) then
    
    Set \( \text{largest so far} \) to \( A_i \)
    
    Set \( \text{location} \) to \( i \)
    
    Add 1 to the value of \( i \)
  
End of the loop
Print out the values of \( \text{largest so far} \) and \( \text{location} \)
Stop

**Figure 2.14** Algorithm to Find the Largest Value in a List

---

**Example 4: Meeting Your Match**

- Pattern matching
  
  - Process of searching for a special pattern of symbols within a larger collection of information
  
  - Is assisting microbiologists and geneticists studying and mapping the *human genome*
Example 4: Meeting Your Match (continued)

• Pattern-matching problem
  
  You will be given some text composed of $n$ characters that will be referred to as $T_1 T_2 \ldots T_n$. You will also be given a pattern of $m$ characters, $m \leq n$, that will be represented as $P_1 P_2 \ldots P_m$. The algorithm must locate every occurrence of the pattern within the text. The output of the algorithm is the location in the text where each match occurred. For this problem, the location of a match is defined to be the index position in the text where the match begins.

Example 4: Meeting Your Match (continued)

• STEP 1
  – The matching process: $T_1 \ T_2 \ T_3 \ T_4 \ T_5 \ldots \ P_1 \ P_2 \ P_3$

• STEP 2
  – The slide forward: $T_1 \ T_2 \ T_3 \ T_4 \ T_5 \ldots$
  – 1-character slide -> $P_1 \ P_2 \ P_3$
Example 4: Meeting Your Match (continued)

- Examples of higher-level constructs
  - Sort the entire list into ascending order
  - Attempt to match the entire pattern against the text
  - Find a root of the equation
- Abstraction
  - Use of high-level instructions during the design process
- Top-down design
  - Viewing an operation at a high level of abstraction
Summary

- Algorithm design
  - A first step in developing an algorithm
- Algorithm design must:
  - Ensure the algorithm is correct
  - Ensure the algorithm is sufficiently efficient
- Pseudocode
  - Used to design and represent algorithms
Summary (continued)

• Pseudocode
  – Readable, unambiguous, and able to be analyzed

• Algorithm design
  – Uses multiple drafts and top-down design to develop the best solution

• Abstraction
  – A key tool for good design